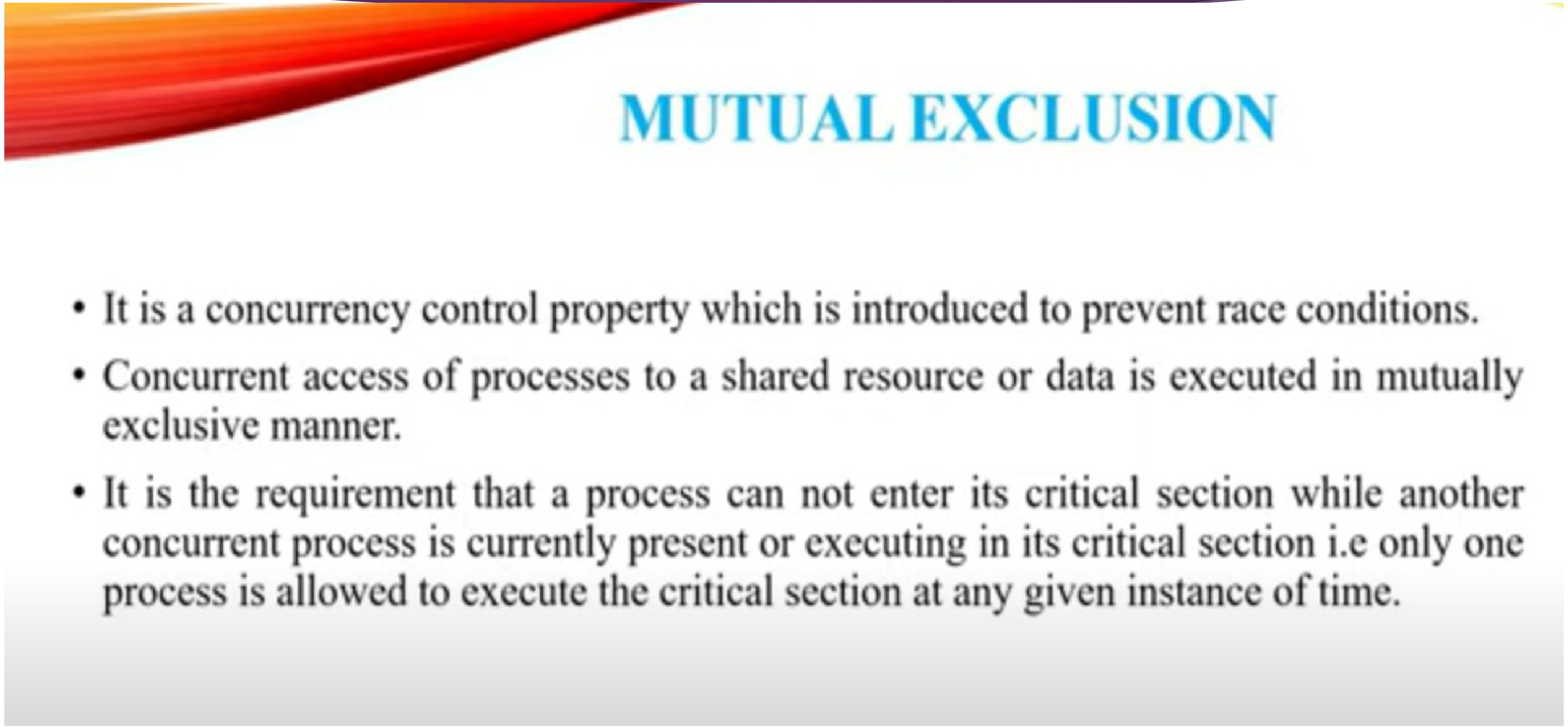
Distributed Mutual Exclusion:-Module-2

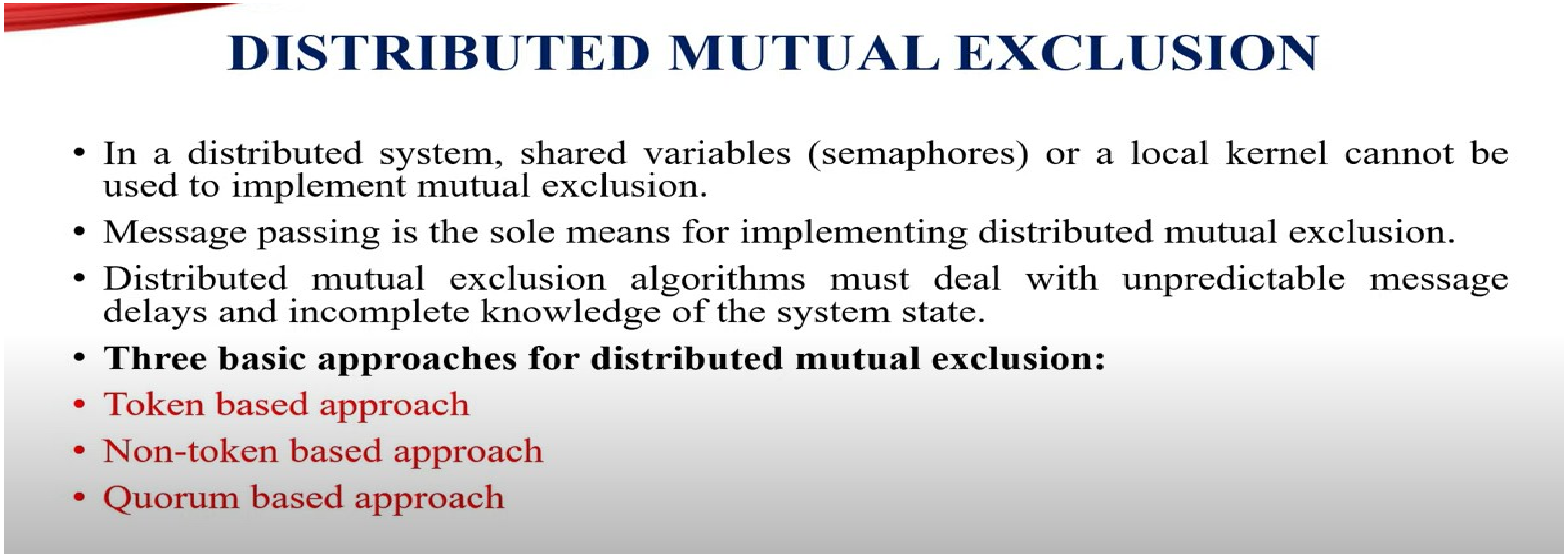


MUTUAL EXCLUSION

- It is a concurrency control property which is introduced to prevent race conditions.
- Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

MUTUAL EXCLUSION IN SINGLE COMPUTER SYSTEM VS. DISTRIBUTED SYSTEM

- In **single computer system**, memory and other resources are shared between different processes.
- The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.
- In **Distributed systems**, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables.
- To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

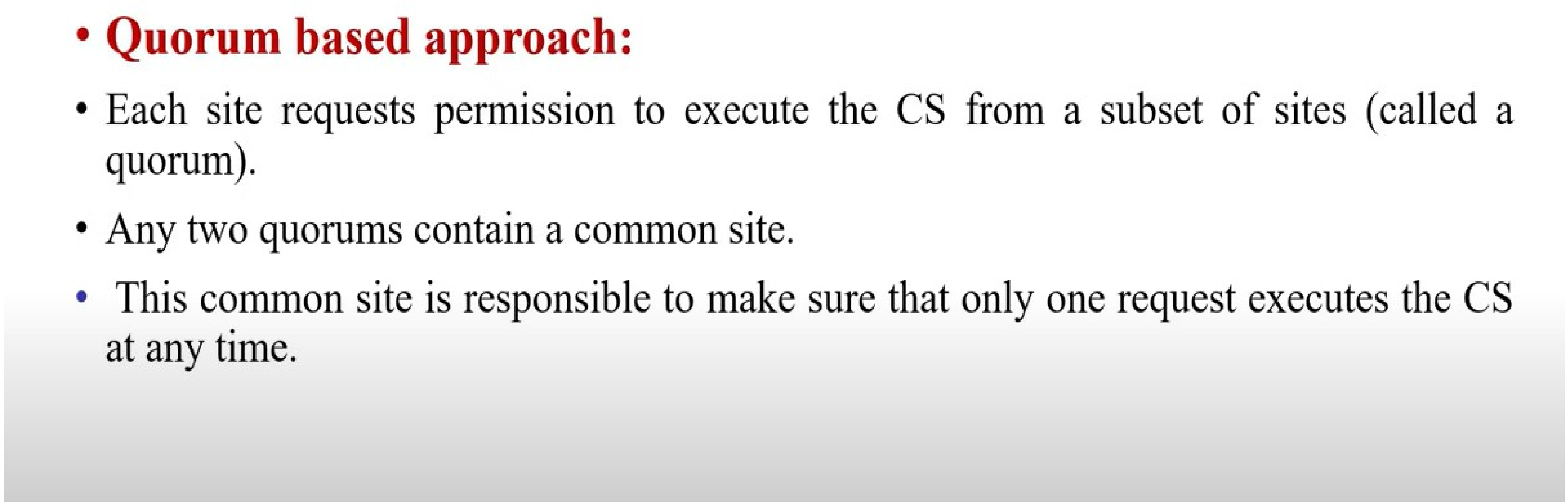


DISTRIBUTED MUTUAL EXCLUSION

- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.
- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.
- **Three basic approaches for distributed mutual exclusion:**
- Token based approach
- Non-token based approach
- Quorum based approach

- **Token-based approach:**
- A unique token is shared among the sites.
- A site is allowed to enter its CS if it possesses the token.
- Mutual exclusion is ensured because the token is unique.

- **Non-token based approach:**
- Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.



- **Quorum based approach:**
- Each site requests permission to execute the CS from a subset of sites (called a quorum).
- Any two quorums contain a common site.
- This common site is responsible to make sure that only one request executes the CS at any time.

Requirements of MUTUAL EXCLUSION Algorithms

- The primary objective of Mutual Exclusion algorithm is to guarantee that only one request access the Critical Section at a time.
- In addition following characteristics are also considered important:

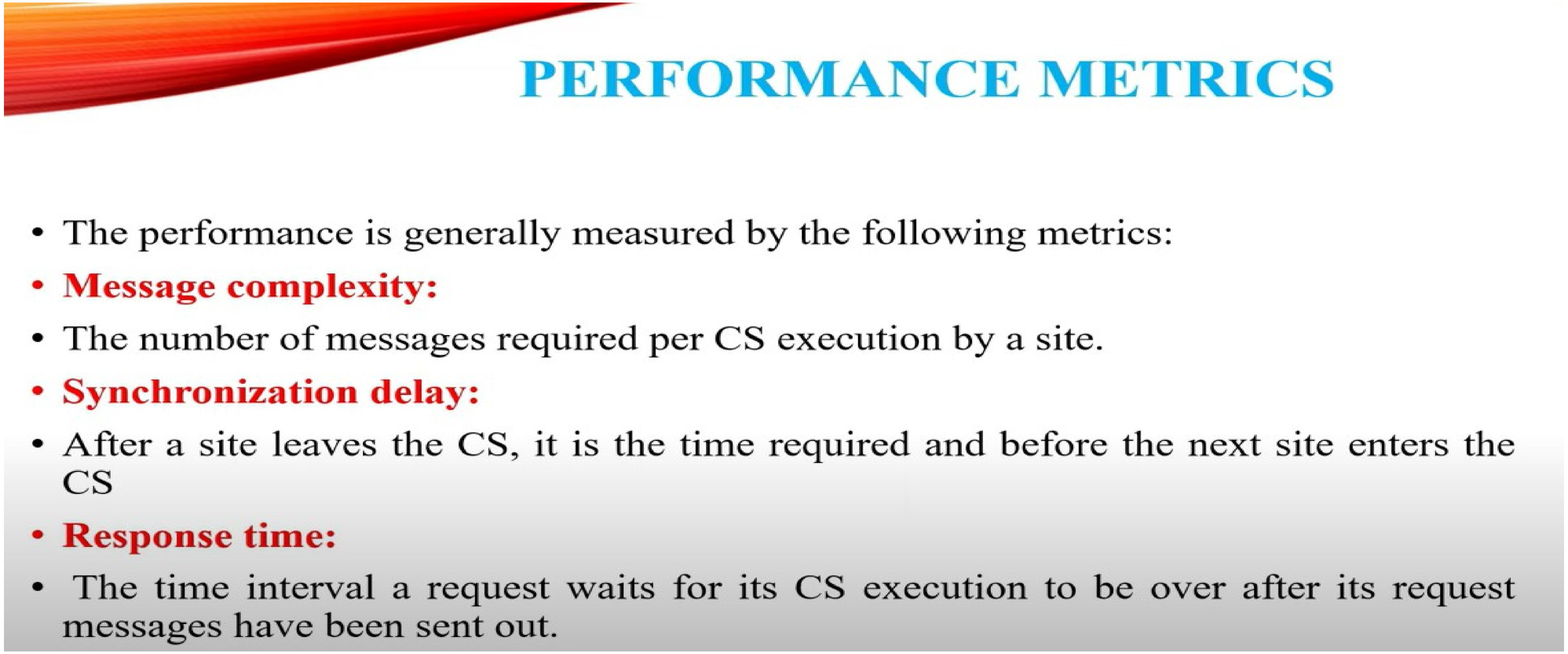
1. Freedom from dead locks: Two or more sites/process should not endlessly wait for messages that will never arrive.
2. Freedom from starvation: A site must not wait indefinitely to execute

the CS while other sites are repeatedly executing the CS.

Requirements of ME Algorithms

3. Fairness: Fairness property generally means that the CS execution requests are executed in order of their arrival in the system.

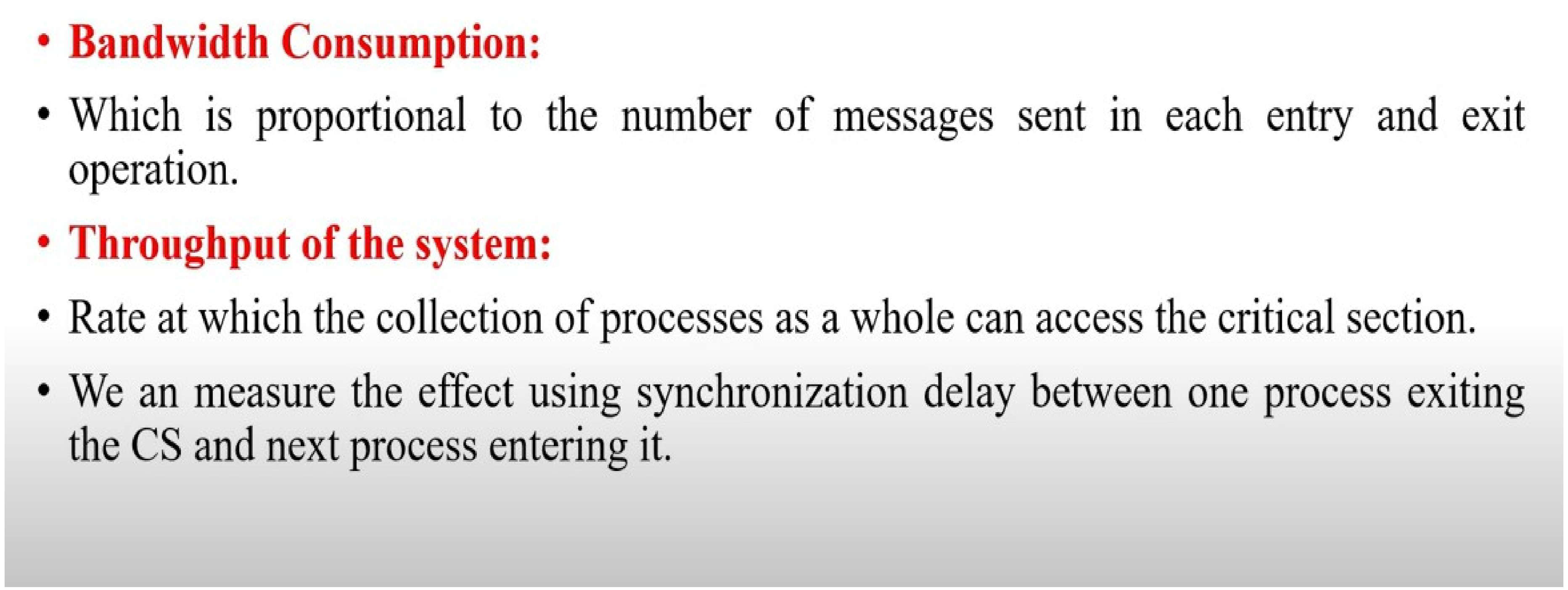
4. Fault Tolerance: In case of a failure, the algorithm can reorganize itself so that it continues to function without any disruptions



PERFORMANCE METRICS

- The performance is generally measured by the following metrics:
- **Message complexity:**
- The number of messages required per CS execution by a site.
- **Synchronization delay:**
- After a site leaves the CS, it is the time required and before the next site enters the CS
- **Response time:**
- The time interval a request waits for its CS execution to be over after its request messages have been sent out.

Performance Matrices

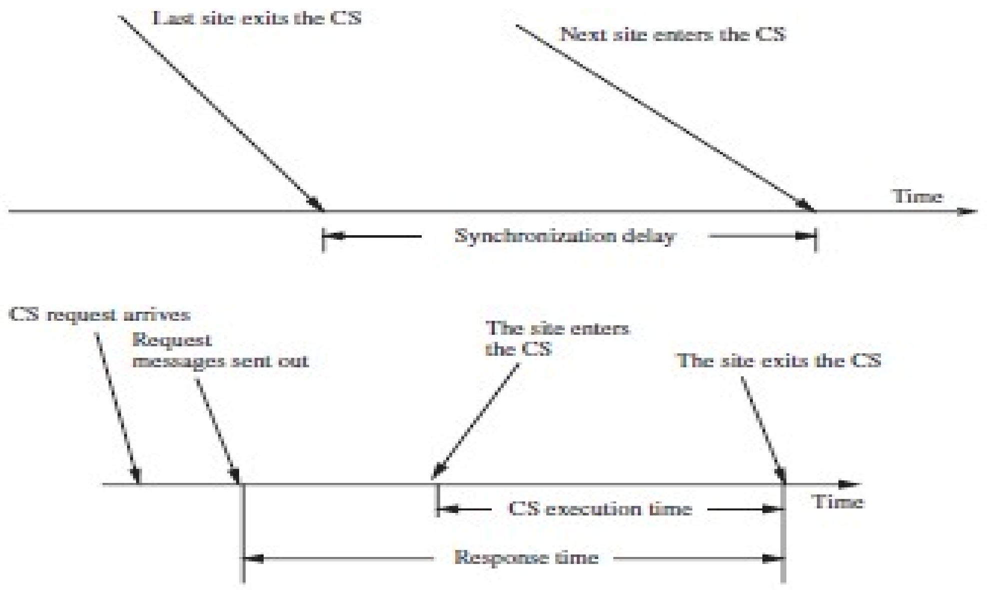


- **Bandwidth Consumption:**
- Which is proportional to the number of messages sent in each entry and exit operation.
- **Throughput of the system:**
- Rate at which the collection of processes as a whole can access the critical section.
- We an measure the effect using synchronization delay between one process exiting the CS and next process entering it.

Performance Matrices

- If **SD** is the synchronization delay and **E** is the average critical section execution time, then the throughput is given by the following equation:

  System throughput = 1/(SD+E)



Low and high load performance

- Performance of a mutual exclusion algorithm depends upon the load.
- Performance of mutual exclusion algorithms are studied under two special loading conditions, “low load” and “high load.”
- Under low load conditions, there is seldom more than one request for the critical section present in the system simultaneously.

Low and high load performance

- Under **heavy load** conditions, there is always a pending request for critical section at a site.
- A site is seldom in the **idle state** in heavy load conditions.
- Performance metrics for Mutual Exclusion algorithms can be easily calculated with mathematical reasoning.

Best and worst case performance

- In the best case, prevailing conditions are such that a performance metric attains the best possible value.
- In most mutual exclusion algorithms the best value of the response time is a roundtrip message delay plus the CS execution time, 2T +E.
- The best and worst cases coincide with low and high loads, respectively.

NON TOKEN BASED ALGORITHMS

- In non token based algorithms a site communicate with a set of other sites to decide who should execute the CS next.
- For a site Si Request set Ri contains ids of all those sites from which site Si must acquire permission before entering the CS
- Non token based algorithms uses time-stamps to order requests for CS and to resolve conflicts between simultaneous requests for the CS
- These algorithms maintain logical clocks and update them according to Lamport’s Scheme.
- Each request for CS gets a timestamp and smaller time stamp requests gets priority over larger time stamp requests

Lamport's Distributed Mutual Exclusion Algorithm

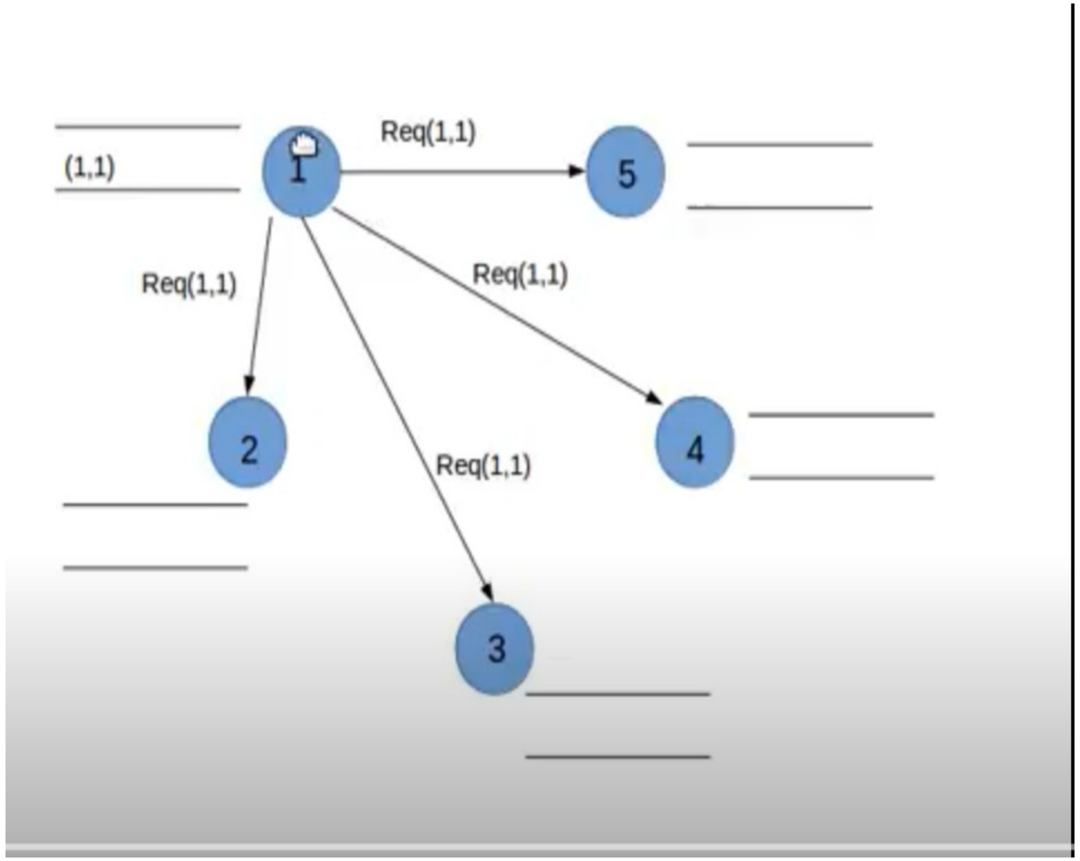
- **Lamport's Distributed Mutual Exclusion Algorithm** is a **permission based** algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.
- **permission based timestamp** is used to order critical section requests and to resolve any conflict between requests.
- In Lamport's Algorithm **critical section requests are executed in the increasing order of timestamps** i.e a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.

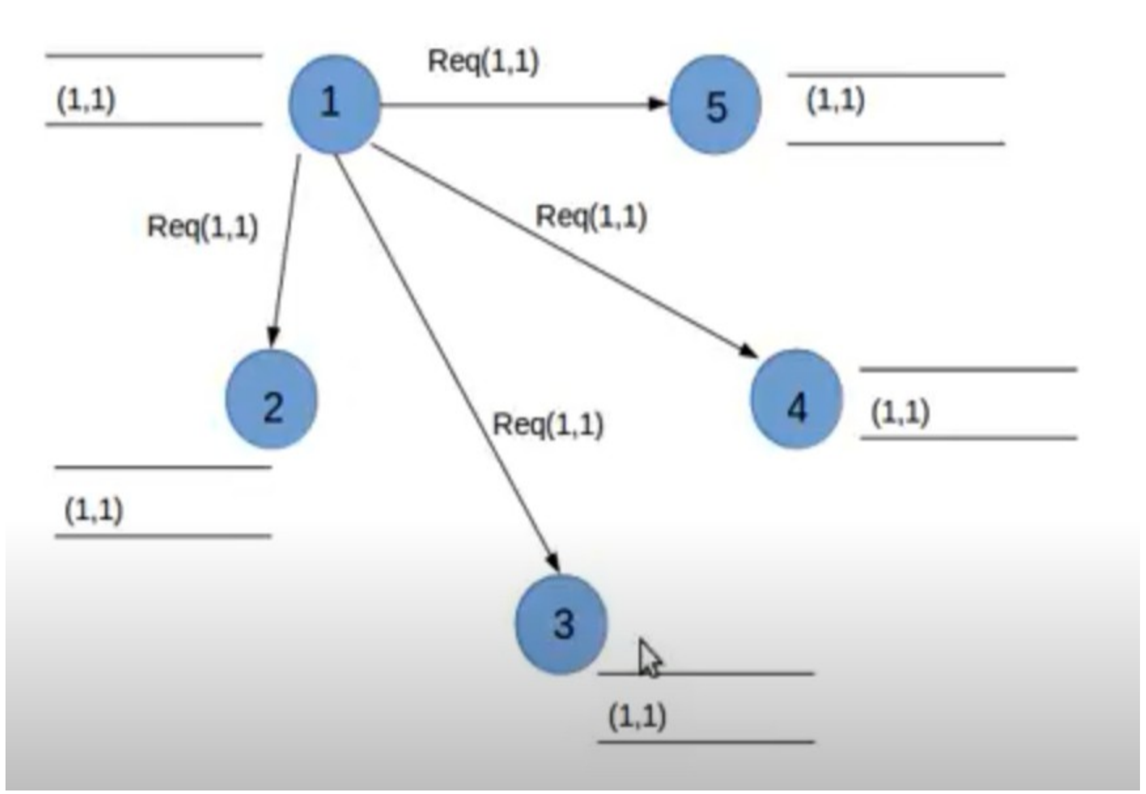
Lamport's Algorithm

In this algorithm

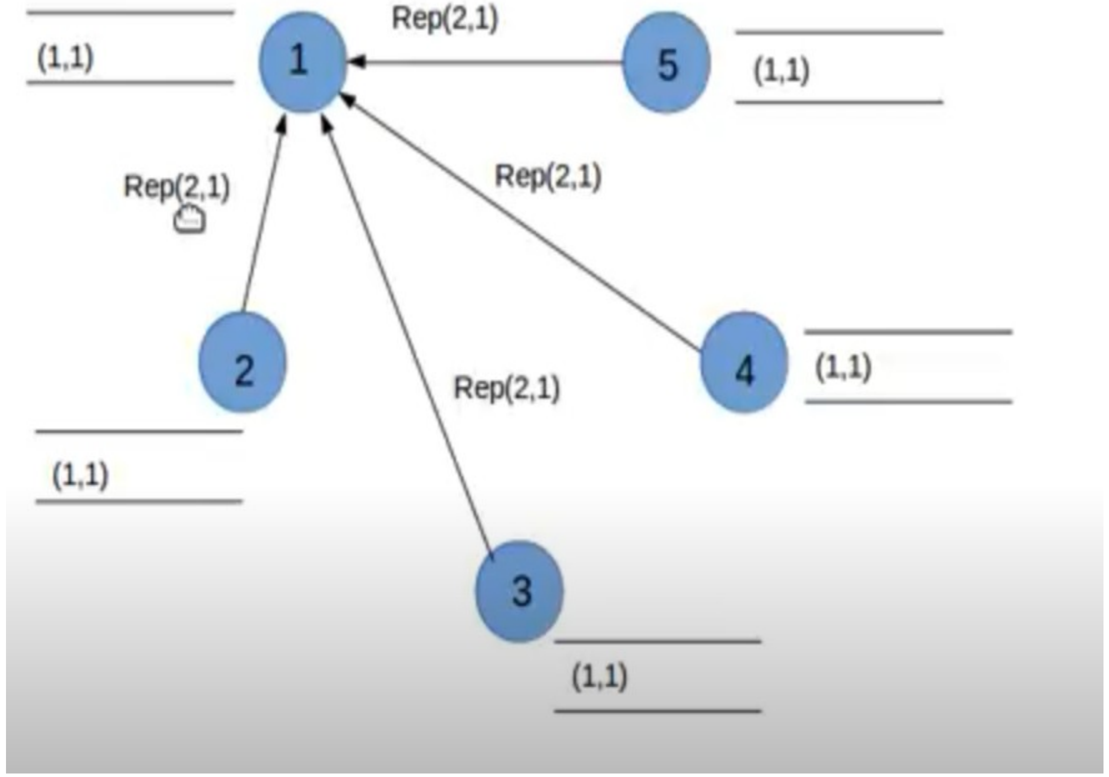
- Three type of messages (**REQUEST**, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
- A site send a **REQUEST** message to all other site to get their permission to enter critical section.
- A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
- A site send a **RELEASE** message to all other site upon exiting the critical section.
- Every site S_i, keeps a queue to store critical section requests ordered by their timestamps.
- **request_queue$_i$** denotes the queue of site S_i
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

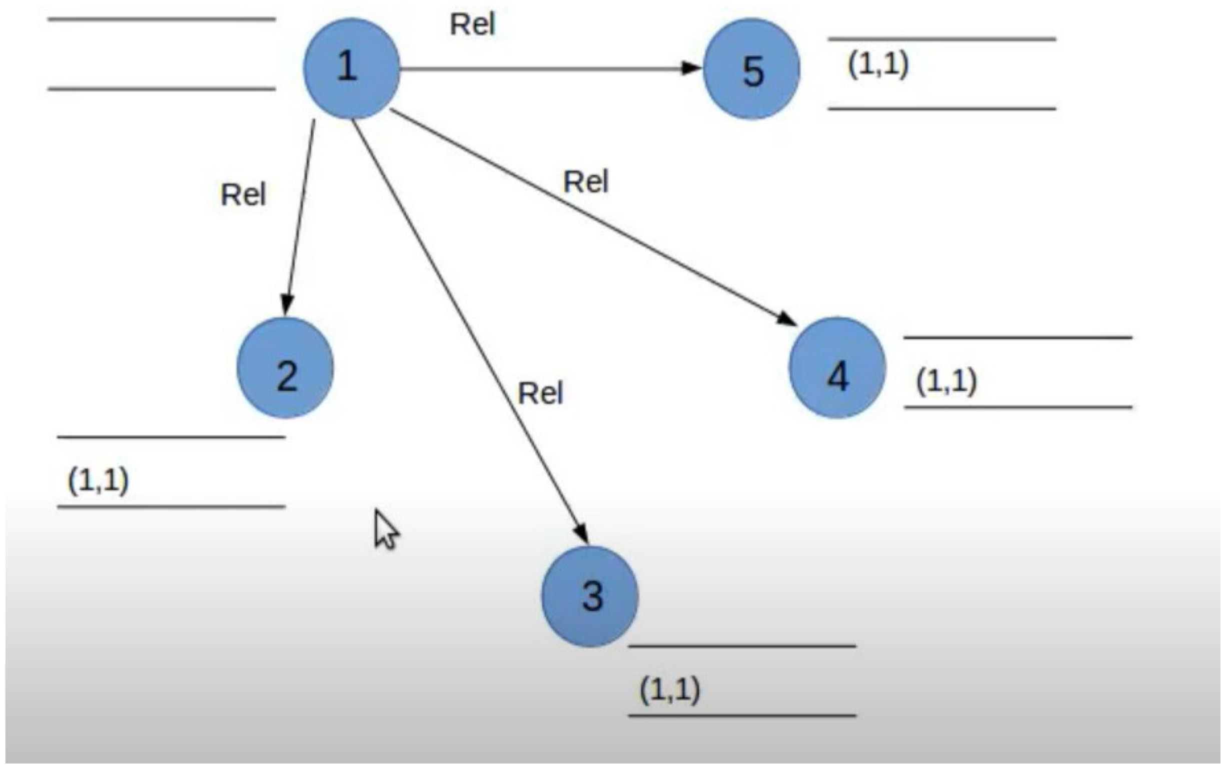
Example

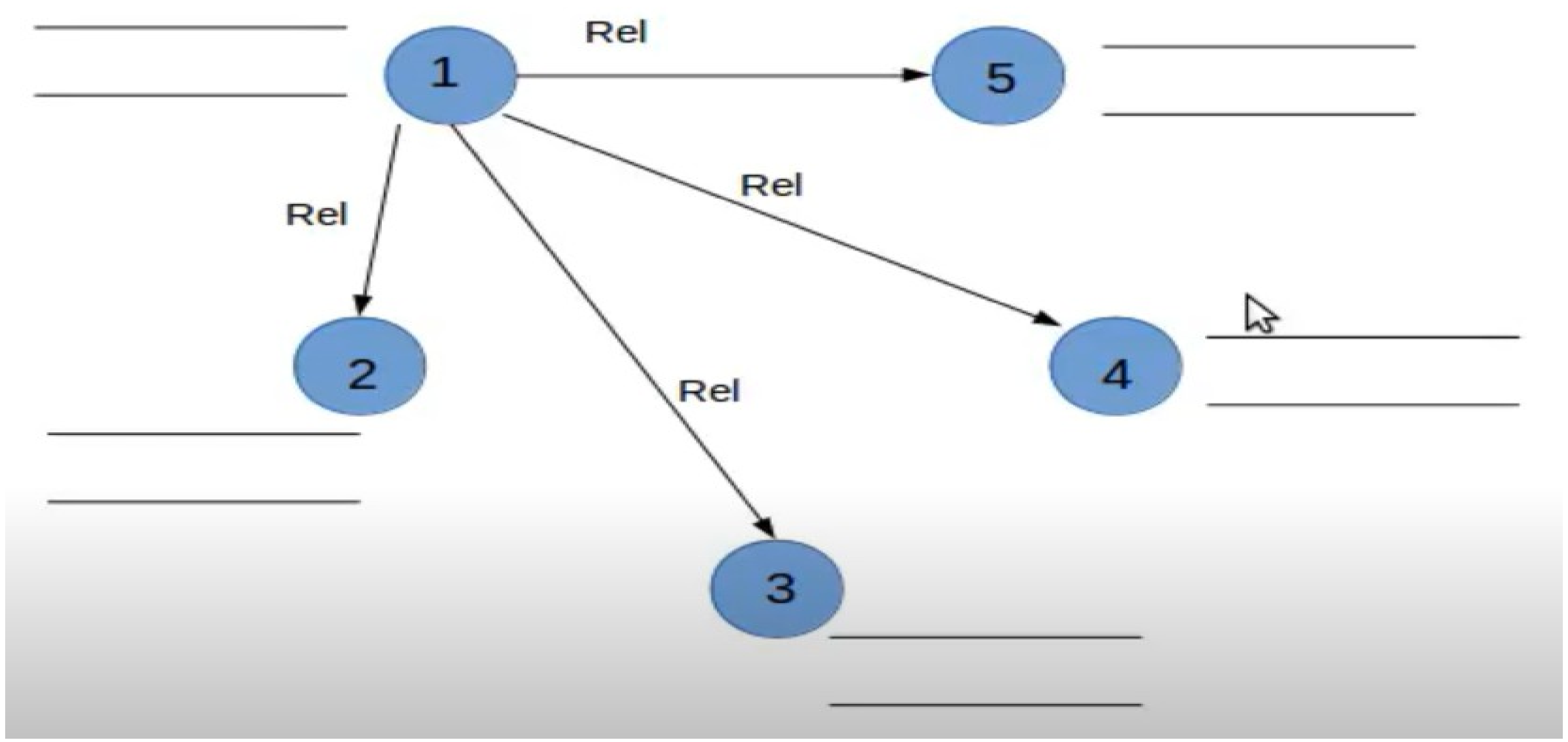

(1,1)
Req(1,1)
Req(1,1)
Req(1,1)
Req(1,1)
1
5
2
4
3


(1,1)
Req(1,1)
(1,1)
Req(1,1)
Req(1,1)
Req(1,1)
1
5
2
4
3
(1,1)
(1,1)
(1,1)

Example- continue






1
Rel
5
Rel
Rel
2
Rel
4
3

Algorithm

To enter Critical section

- When a **site S_i** wants to enter the critical section, it sends a request message **Request(ts_i, i)** to all other sites and places the request on **request_queue$_i$**. Here, Ts_i denotes the timestamp of Site S_i.
- When a **site S_j** receives the request message **REQUEST(ts_i, i)** from site S_i, it returns a timestamped REPLY message to site S_i and places the request of site S_i on **request_queue$_j$**

To execute the critical section

- Site Si enters the CS when the following two conditions hold:

L1: Si has received a message with timestamp larger than (tsi, i) from all other sites.

L2: Si's request is at the top of request_queuei

Algorithm

To release the critical section:

- Site **Si**, upon **exiting** the CS, removes its request from the top of its **request queue** and broadcasts a timestamped **RELEASE** message to all other sites.

- When a site **Sj** receives a **RELEASE** message from site Si, it removes Si's request from its **request queue**.

- When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

Lamport’s algorithm: Performance

- For each CS execution, Lamport’s algorithm requires N −1 REQUEST messages, N −1 REPLY messages, and N −1 RELEASE messages.
- Ie, it requires 3(N-1) messages per CS invocation.
- The Lamport’s Algorithm can be optimized by reducing the no:of message to lie between 3(N-1) and 2(N-1).
- This can be achieved by supressing REPLY messages in certain situations

Ricart–Agrawala Algorithm in Mutual Exclusion in Distributed System

- **Ricart–Agrawala algorithm** is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport’s Distributed Mutual Exclusion Algorithm. Like Lamport’s Algorithm, it also follows permission based approach to ensure mutual exclusion.
- The Ricart–Agrawala algorithm assumes that the communication channels are **FIFO**.
- **The algorithm uses two types of messages:**
  - REQUEST
  - REPLY.

Ricart–Agrawala Algorithm

- A process sends a **REQUEST message** to all other processes to request their permission to enter the critical section.
- A process sends a **REPLY message** to a process to give its permission to that process.
- Lamport logical clocks to assign a **timestamp** to critical section requests.
- Timestamps are used to decide the priority of requests in case of conflict. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.
- In this algorithm ,for every requesting site, the site with higher priority(smaller timestamp) will always defer the request of the lower priority site.
- So the process with **high priority gets** to execute the CS

Ricart–Agrawala algorithm- Requesting critical section

- When a site S_i wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.
- When a site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if and only if
  - Site S_j is neither requesting nor currently executing the critical section.
  - In case Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request.Otherwise the request is deferred by site S_j.

Ricart–Agrawala algorithm- Executing and Releasing critical section

- Site S_i enters the critical section if it has received the **REPLY** message from all other sites.

Releasing the critical section.

- When site Si exits the CS, it sends all the REPLY messages to all deferred requests.
- The site with next highest priority request receives the last needed REPLY message and enters the CS

Ricart–Agrawala algorithm- Performance

- For each CS execution, the Ricart–Agrawala algorithm requires $N - 1$ REQUEST messages and $N - 1$ REPLY messages.

- Thus, it requires $2(N - 1)$ messages per CS execution.

TOKEN-BASED algorithm

- A unique token is shared among all sites
- A site is allowed to enter its critical session if it possesses the token
- Token based algorithms uses sequence numbers instead of time stamps
- Every request for token contains a sequence number and sequence numbers of sites advance independently.
- A site increments its sequence number counter every time it makes a request for token
- Primary function of sequence number is to distinguish b/w old and current request for token

SUZUKI –KASami’s Broadcast algorithm

- If a site attempting to enter a CS does not have the token, it broadcasts a REQUEST message for the token to all other sites.
- A site that possesses the token sends it to the site that sends the REQUEST message.
- If the site possessing the token is executing the CS, it sends the token only after it has exited the CS.
- A site holding the token can repeatedly enter the critical session until it sends the token to some other site.

SUZUKI –KASami’s Broadcast algorithm

- The main design issues in this algorithm are:
- How to distinguishing an outdated REQUEST message from a current REQUEST message.
- How to determine which site has an outstanding request for the CS

SUZUKI –KASami's Broadcast algorithm

- Outdated REQUEST messages are distinguished from current REQUEST messages in the following manner:
- A REQUEST message of site Sj has the form REQUEST(j, n) where n (n = 1 2) is a sequence number that indicates that site Sj is requesting its nth CS execution.
- A site Si keeps an array of integers RNi[1, … ,N] where RNi[j] denotes the largest sequence number received in a REQUEST message so far from site S.
- When site Si receives a REQUEST(j, n) message, it sets RNi[j]= *max*(RNi[j], n).

SUZUKI –KASami's Broadcast algorithm

- When a site Si receives a REQUEST(j, n) message, the request is outdated if RNi[j]> n.
- Sites with outstanding requests for the CS are determined in the following manner:
- the token consists of a queue of requesting sites, Q, and an array of integers LN[1, … ,N], where LN[j] is the sequence number of the request which site Sj executed most recently.
- After executing its CS, a site Si updates LN[i] : = RNi[i] to indicate that its request corresponding to sequence number RNi[i] has been executed.
- Token array LN[1, … ,N] permits a site to determine if a site has an outstanding request for the CS.

SUZUKI –KASami’s Broadcast algorithm

- In Site Si if RNi[j]=LN[j]+1, then site Sj is currently requesting a token.
- After executing the CS, a site checks this condition for all the j’s to determine all the sites that are requesting the token and places their i.d.’s in queue Q if these i.d.’s are not already present in Q.
- Finally the site sends the token to the site whose i.d. is at the head of Q.

Requesting the critical section:

- If requesting site Si does not have the token, then it increments its sequence number, RNi[i], and sends a REQUEST(i, sn) message to all other sites. *("sn" is the updated value of RNi[i])*

- When a site Sj receives this message, it sets RNj[i] to *max*(RNj[i], sn). If Sj has the idle token, then it sends the token to Si if RNj[i]=LN[i]+1.

Executing and releasing the critical section:

Executing CS

- Site Si executes the CS after it has received the token.

Releasing the CS

Having finished the execution of the CS, site Si takes the following actions:

- It sets LN[i] element of the token array equal to RNi[i].
- For every site Sj whose i.d. is not in the token queue, it appends its i.d. to the token queue if RNi[j] = LN[j]+1.
- If the token queue is nonempty after the above update, Si deletes the top site i.d. from the token queue and sends the token to the site indicated

Potential Security Violations

Three categories of potential security violations are:

Unauthorized information release: This occurs when an unauthorized person is able to read and take advantage of the information stored in a computer system. This also includes the unauthorized use of a computer program.

Unauthorized information modification: This occurs when an unauthorized person is able to alter the information stored in a computer. Examples include changing student grades in a university database and changing account balances in a bank database.

Unauthorized denial of service: An unauthorized person should not succeed in preventing an authorized user from accessing the information stored in a computer.

External Vs. Internal Security

Computer system security can be divided into ***external security*(physical) *& internal security.***

Design Principles for Secure Systems

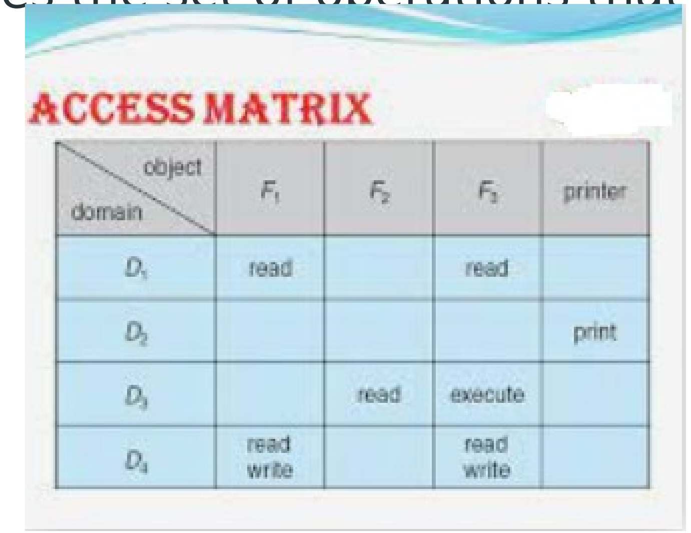
- **Security Design Principles**

- **Economy of Mechanism-** A protection mechanism should be economical to develop and use
- **Complete Mediation-** The design of a completely secure system requires that every request to access an object be checked for the authority to do so.
- **Open Design-** The Open Design Design Principle is a concept that the security of a system and its algorithms should not be dependent on secrecy of its design or implementation.(A protection mechanism should work even if its underlying principles are known to an attacker.)
- **Separation of Privileges-** A protection mechanism that requires two keys   to unlock a lock(or gain access to a protected object) is more robust and flexible than one that allows only a single key to unlock a lock.
- **Least Privilege-** A subject should be given the bare minimum access rights that are sufficient for it to complete its task.
- **Least Common Mechanism -** The Least Common Mechanism design principle declares that mechanisms used to access resources should not be shared.
- **Acceptability-** A protection mechanism must be simple to use. A complex and obscure protection mechanism will deter users from using it.
- **Fail-Safe Defaults-** The Fail-Safe Defaults design principle pertains to allowing access to resources based on granted access over access exclusion. This principle is a methodology for allowing resources to be accessed only if explicit access is granted to a user. By default users do not have access to any resources until access has been granted. This approach prevents unauthorized users from gaining access to resource until access is given.

The Access Matrix Model

Access Matrix is a security model of protection state in computer system. It is represented as a matrix. Access matrix is used to define the rights of each process executing in the domain with respect to each object. The rows of matrix represent domains and columns represent objects.

Each cell of matrix represents set of access rights which are given to the processes of domain means each entry(i, j) defines the set of operations that a process executing in domain Di can invoke on object Oj.



ACCESS MATRIX

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Implementation of Access Matrix

- **Access matrix** is likely to be very sparse and takes up a large chunk of memory. Therefore direct implementation of access matrix for access control is storage inefficient.
- The inefficiency can be removed by decomposing the access matrix into rows or columns.Rows can be collapsed by deleting null values and so for the columns to increase efficiency. From these approaches of decomposition **three implementation of access matrix** can be formed which are widely used. They are as follows:

  1. Capabilities

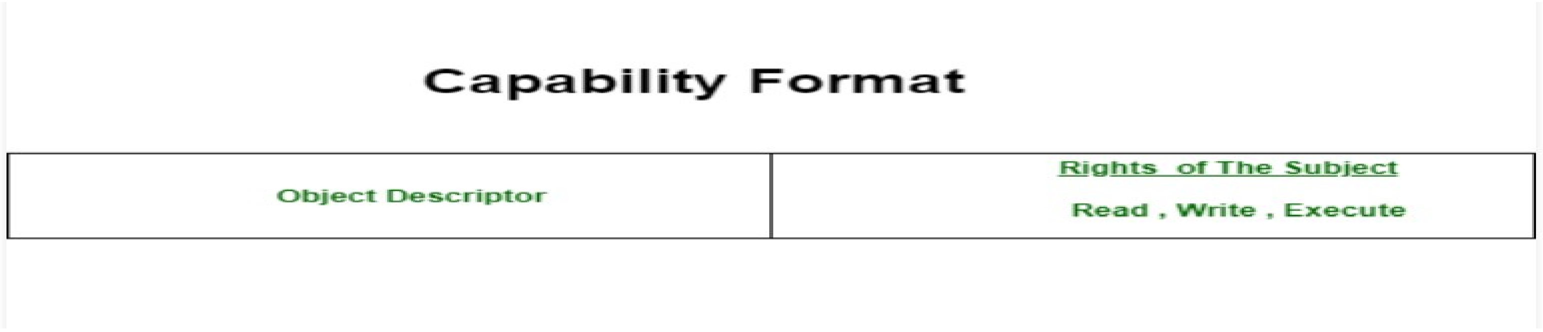
  2. Access Control List

  3. Lock and Key Method

Capabilities:

- **Capabilities:**
  This method refers to **row wise decomposition of the access matrix**. Each Subject is assigned with a list of tuples *(o, M[s, o])* for all objects o that it is allowed to access. This **tuples are called Capabilities**. If a subject possess a capability (o, M[s, o]) then it is allowed to access object o in the manner which is described in M[s, o]. A subject is allowed to access any objects for which it holds the capabilities.Capabilities are not meant to be forged.Capabilities contain two fields:

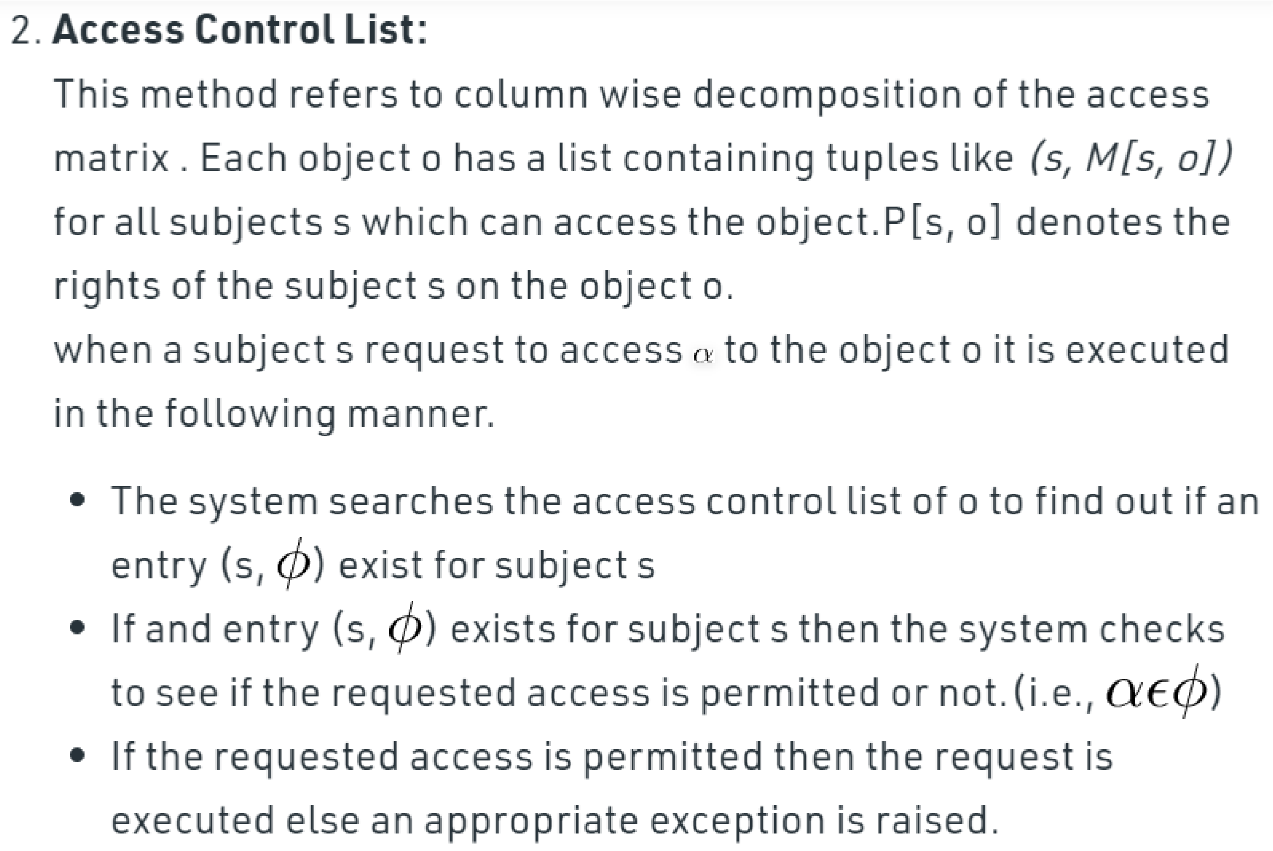
  1. Object Descriptor.



Capability Format

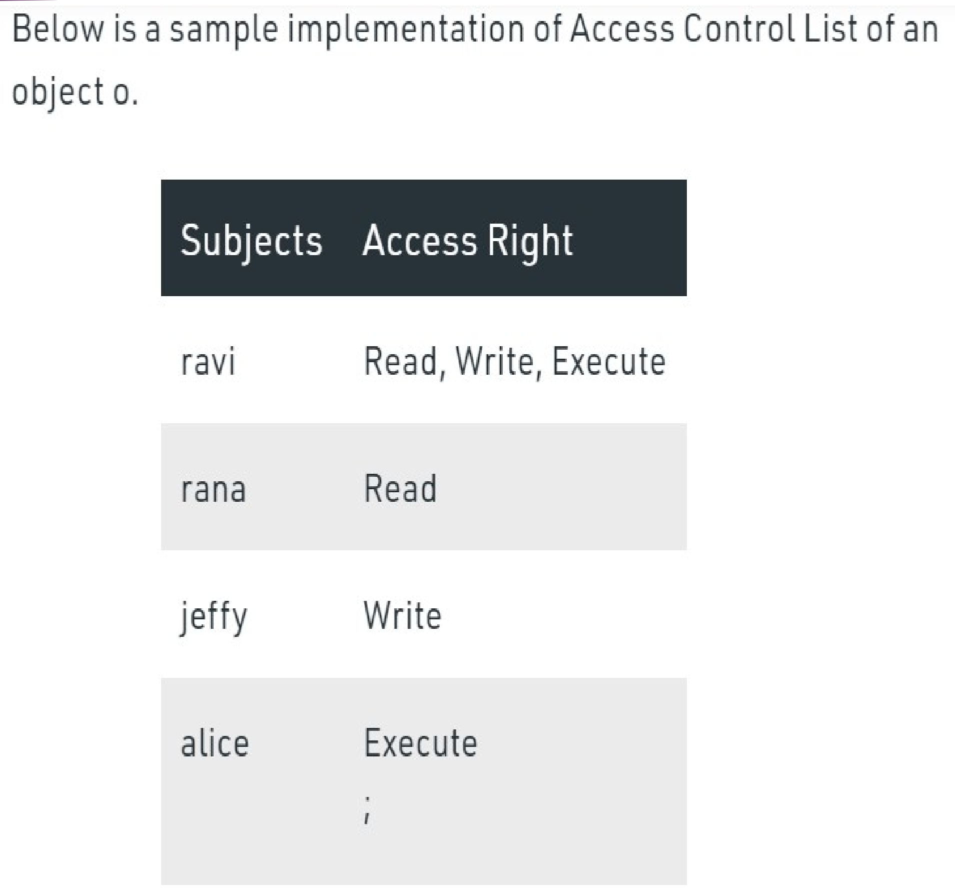
Object Descriptor	Rights of The Subject Read , Write , Execute

Access Control List:



2. **Access Control List:**
This method refers to column wise decomposition of the access matrix . Each object o has a list containing tuples like *(s, M[s, o])* for all subjects s which can access the object.P[s, o] denotes the rights of the subject s on the object o.
when a subject s request to access α to the object o it is executed in the following manner.

- The system searches the access control list of o to find out if an entry (s, ϕ) exist for subject s
- If and entry (s, ϕ) exists for subject s then the system checks to see if the requested access is permitted or not.(i.e., $\alpha \epsilon \phi$)
- If the requested access is permitted then the request is executed else an appropriate exception is raised.



Below is a sample implementation of Access Control List of an object o.

Subjects	Access Right
ravi	Read, Write, Execute
rana	Read
jeffy	Write
alice	Execute ;

Easy revocation and **Easy review of an access** are the major feature of access control list.